Pascal Pons

Follow

SOLVING CONNECT FOUR

Part 12 – Lower bound transposition table

When a node is not pruned by alpha-beta, we get an upper-bound of the score at the end of negamax function. So far we are storing this upper bound in our transposition table.

However, when the exploration is pruned by an alpha-beta cut, the returned score is a lower bound. Keeping and reusing these lower bound scores in a transposition table can help reducing the number of explored nodes.

however, pruned nodes are less expensive to explore than nodes without pruning (as we only partially explore pruned nodes while we fully explore non-pruned node). The expected gain is thus smaller than upper bound transposition tables.

Keeping both upper and lower bound in the same transposition table.

We want to keep both upper and lower bounds in transposition tables. It is possible to instanciate two transposition tables, one for upper bounds and one for lower bounds. It is also possible to store them in the same table while adding a flag in the stored value to differentiate upper and lower bounds. In practice, it appears that this second approach is more efficient for the same overall storage size.

We will thus store both upper and lower bounds in the same transposition table. We keep storing the same value for upper bounds and we shift (add a constant ofset) the lower bound values by the max possible score. the number of possible values is doubled and is now 2*(max_score - min_score + 1). We also have to give one additional bit of storage to each transposition table values.

The code is updated as below to store the new lower bounds in the alpha-beta pruning loop. Note the new transposition table insertion right before returning eary score when score >= beta.

```
    while(uint64_t next = moves.getNext()) {
        Position P2(P);
        P2.play(next);  // It's opponent turn in P2 position after current player plays x column.
        int score = -negamax(P2, -beta, -alpha); // explore opponent's score within [-beta;-alpha] windows:
        // no need to have good precision for score better than beta (opponent's score worse than -beta)
        // no need to check for score worse than alpha (opponent's score worse better than -alpha)

        if(score >= beta) {
          transTable.put(key, score + Position::MAX_SCORE - 2*Position::MIN_SCORE + 2); // save the lower bound of the position
          return score;  // prune the exploration if we find a possible move better than what we were looking for.
        }
        if(score > alpha) alpha = score; // reduce the [alpha;beta] window for next exploration, as we only
        // need to search for a position that is better than the best so far.
      }
```

And this is how we now fetch the cached scores, updating respectively alpha or beta depending on wether we retreived a lower or an upper bound.

```
const uint64_t key = P.key();

if(int val = transTable.get(key)) {    // fetch potential stored lower or upper bound of the score
  if(val > Position::MAX_SCORE - Position::MIN_SCORE + 1) { // we have an lower bound
    min = val + 2*Position::MIN_SCORE - Position::MAX_SCORE - 2;
    if(alpha < min) {
      alpha = min;                     // there is no need to keep alpha below our min possible score.
      if(alpha >= beta) return alpha;  // prune the exploration if the [alpha;beta] window is empty.
    }
  }
  else { // we have an upper bound
    max = val + Position::MIN_SCORE - 1;
    if(beta > max) {
      beta = max;                     // there is no need to keep beta above our max possible score.
      if(alpha >= beta) return beta;  // prune the exploration if the [alpha;beta] window is empty.
    }
  }
}
```

Full source code corresponding to this part.

Benchmark

Storing lower bounds in the transposition table allows to prune a little more the exploration using the same size of Transosition Table. We reduce the number of explored nodes by 10% to 15% but the overall computational improvement is smaller:

Solver	Test Set name	mean time	mean nb pos	K pos/s
Lower bound transposition table (strong solver)	End-Easy	4.568 µs	51.28	11,230
Lower bound transposition table (strong solver)	Middle-Easy	37.45 µs	449.6	12,000
Lower bound transposition table (strong solver)	Middle-Medium	3.212 ms	39,900	12,420

Solver	Test Set name	mean time	mean nb pos	K pos/s
Lower bound transposition table (strong solver)	Begin-Easy	254.6 µs	3,298	12,950
Lower bound transposition table (strong solver)	Begin-Medium	96.63 ms	1,201,000	12,430
Lower bound transposition table (strong solver)	Begin-Hard	5.490 s	65,920,000	12,010
Lower bound transposition table (weak solver)	End-Easy	3.217 µs	29.36	9,125
Lower bound transposition table (weak solver)	Middle-Easy	47.73 µs	532.7	11,160
Lower bound transposition table (weak solver)	Middle-Medium	1.717 ms	20,210	11,760
Lower bound transposition table (weak solver)	Begin-Easy	1.870 ms	22,230	11,890
Lower bound transposition table (weak solver)	Begin-Medium	45.01 ms	538,700	11,970
Lower bound transposition table (weak solver)	Begin-Hard	3.149 s	38,120,000	12,110

Tutorial plan

SOLVING CONNECT FOUR

Updated: January 12, 2019

Previous | Next